TAUPE: Towards Understanding Program Comprehension

Yann-Gaël Guéhéneuc
PTIDEJ Team – LaiGLE
Département d'informatique et de recherche opérationnelle
Université de Montréal – CP 6128 succ. Centre Ville
Montréal, Québec, H3C 3J7 – Canada
guehene@iro.umontreal.ca

Abstract

Program comprehension is a very important activity during the development and the maintenance of programs. This activity has been actively studied in the past decades to present software engineers with the most accurate and—hopefully—most useful pieces of information on the organisation, algorithms, executions, evolution, and documentation of a program. Yet, only few work tried *to understand concretely how software engineers obtain and use this information*. Software engineers mainly use *sight* to obtain information about a program, usually from source code or class diagrams. Therefore, we use eye-tracking to collect data about the use of class diagrams by software engineers during program comprehension. We introduce a new visualisation technique to aggregate and to present the collected data. We also report the results and surprising insights gained from two case studies.

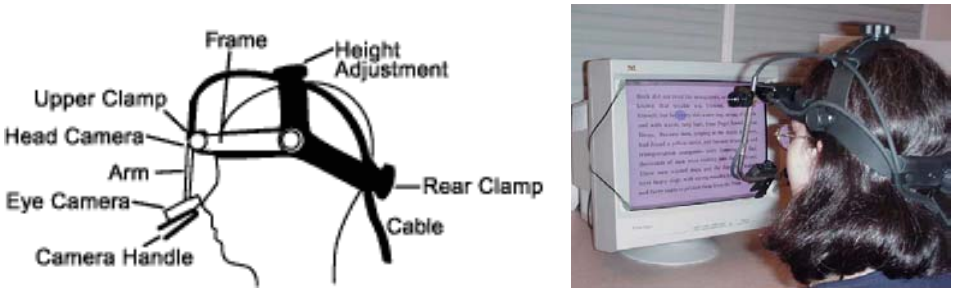
1 Introduction

Program comprehension is one of the most important activity during the development and maintenance of programs. This activity is "necessary to facilitate reuse, inspection, maintenance, reverse engineering, re-engineering, migration, and extension of existing software systems" [11]. It consists, for a software engineer, in acquiring information about the organisation, algorithms, executions, evolution, and documentation of a program to form an accurate mental model of this program. This mental model allows software engineers to perform appropriate changes without introducing bugs [23] or degrading the program design [18].

Previous work analysed the activity of program comprehension and suggested techniques to ease this activity. This work includes understanding the creation of mental models [23], identifying interesting information [13], proposing new techniques to present information [22]. However, to the best of our knowledge, no previous work studied *how software engineers concretely acquire information about a program*. The acquisition of information is mainly performed through *sight*, from a computer screen. Thus, we propose to study the use of sight by software engineers during program comprehension to better understand this activity. A better understanding of the acquisition of information will *help in combining existing techniques and in developing new techniques to ease program comprehension*.

UML [15] has become *de facto* the standard notation to describe the organisation, structure, and behaviour of object-oriented programs. In particular, class diagrams describing the organisation of classes, their relationships and their services, are often used by software engineers to obtain information on the design of objet-oriented programs [6, 19, 24].

We use eye-tracking to collect data on the acquisition of information from class diagrams by software engineers through sight to understand concretely how software engineers obtain information and use this source of information.



(a) Headband and its different parts.

(b) Use of the headband.

Figure 1: SR Research EyeLink II eye-tracking system.

The data collected using eye-tracking consists of two types of particularly useful information: fixations and saccades. Fixations record the position of the eye during a gaze while saccades record the movements of the eyes between two fixations. The numbers of fixations and saccades can vary from a dozen to several hundreds depending on the size and complexity of the class diagram, on the time spent looking at the diagram, on the strategy used to acquire information from a diagram, and on the purpose of the activity of program comprehension at hand.

No two software engineers analyse a same diagram identically. Therefore, we develop a new visualisation technique to aggregate and to present the fixations and saccades collected from several software engineers. Our technique introduces the concept of areas of interest and superimposes aggregations of the fixations and saccades and a class diagram with alpha composition to highlight the most visited areas of the diagram. We implement our tool as the Taupe data viewer. We perform two case studies with two different class diagrams and program comprehension activities with software engineers and report results using our visualisation technique. These case studies show that, surprisingly, binary class relationships (inheritance, use, association, aggregation, and composition [7]) seem to be scarcely used. Thus, our contributions are:

- The first use of eye-trackers to collect data on the concrete use of class diagrams during program comprehension.
- A new visualisation technique to aggregate and to present collected data and its implementation, the Taupe data viewer.
- Two case studies of the analysis of program comprehension with class diagrams performed with software engineers.

Section 2 introduce the idea of using eye-tracking to study the comprehension of class diagrams. Section 3 present our visualisation technique and its implementation, the Taupe data viewer. Section 4 report the results of two first case studies with class diagrams. Finally, Section 5 concludes and presents future work.

2 Class Diagrams and Eye-Tracking

Class diagrams represent the structure and global behaviour [10] of programs, showing classes, interfaces, and their relationships. They are often used by software engineers during development and maintenance to abstract implementation details and to present an easier-to-grasp clustered view of the program source code [6, 19, 24].

2.1 Previous Work

Class diagrams have been extensively studied in the literature on program comprehension. We only summarise here some of the main lines of research on program comprehension using class diagrams.

Purchase *et al.* [1] reported the results of experiments on the effect of aesthetics criteria on the preferences of users for UML class diagrams. They performed several experiments with subjects to assess the subjects' preferences over several pairs of class diagrams, each diagram in a pair conforming to different (but related) aesthetics criteria. They collected quantitative data in the form of percentages of subjects preferring one diagram over another in each pair and qualitative assessments of each diagram. They concluded on the most important aesthetic criteria for UML class diagrams, including joined inheritance arcs and directional indicators.

Eichelberger [3] studied the relation between the UML notation for class diagrams, principles of human–computer interactions, and principles of object-oriented design and programming. The author then suggested changes to the UML notation and aesthetics criteria to lay out class diagrams. These changes and aesthetic criteria are implemented in a tool, SUGIBIB, to lay out UML-like class diagrams. The author claimed that laying out class diagrams while conforming to the aesthetics criteria improve the readability of the diagrams but he only provides qualitative arguments.

Hadar and Hazzan [8] presented results from a study on the startegies applied by software engineers in the process of comprehending visual models of programs. They use visual models that were described using the UML notation and included use case, activity, class, sequence, collaboration, state chart, object, package, and deployment diagrams. The subjects were senior students majoring in computer science from several universities. The subjects were divided in two groups to collect both qualitative and quantitative data. The authors concluded on the usefulness of multifaceted descriptions of programs provided by the UML and that no one type of diagram was more important than the other. However, further studies should be performed to confirm these findings.

Sun and Wong [24] evaluated the layout algorithms for class diagrams of two industrial tools, Rational Rose and Borland Together, according to criteria from previous work, including the cited works by Purchase [1] and Eichelberger [3]. Using laws from the Gestalt theory of visual perception [16, p. 50–53], they retained and justified 14 criteria to assess the visual quality of the layouts of class diagrams. They applied these criteria on a Thermometer program and on JUNIT [4]. They concluded on the good quality of both industrial tools, on the relevance of their criteria, and on the difficulty of satisfying all criteria.

Previous work developed and studied criteria and techniques to ease program comprehension in general and using class diagrams in particular. Yet, to the best of our knowledge, no previous work studied the *concrete acquisition of information from class diagrams*.

2.2 General Setting

Progress in non-intrusive monitoring of human behaviour allows studying the external behaviour of software engineers involved in program comprehension activities while disturbing their activities as little as possible. In particular, the use of video-based eye tracking systems allows recording software engineers' eye movements when they look at a class diagram to understand the modelled program.

A video-based eye tracking system collects data on the relative coordinates of a subject's eye movements over a computer screen, through a special headband, without interfering with the subject's activity, if the subject keeps a relatively steady position in front of the screen. The system converts the collected raw coordinates in fixations and saccades using thresholds based on human physiology.

We use the eye-tracking systems provided by SR Research to study software engineers during program comprehension. SR Research is an international manufacturer of high quality eye-tracking systems. It provides the EyeLink II system, which decomposes in a display computer displaying the data to be analysed by a software engineer, a host computer storing the data on an engineer's eye movements, and a headband supporting cameras to track the eye movements.

Figure 1(a) depicts the headband and its different parts. The headband mainly consists of a set of cameras recording the position of the head and the eye movements with respect to the image displayed on the computer screen:

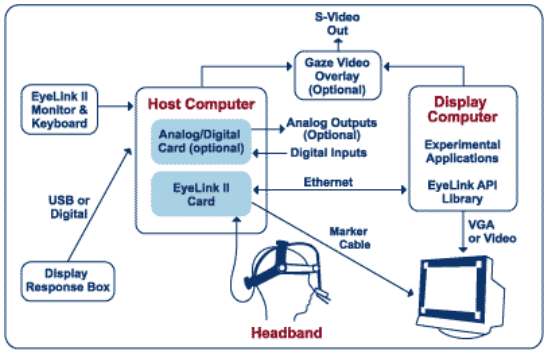


Figure 2: Use of a EyeLink II system.

active components are the eye cameras and head camera, which capture the eye movements and relate the position of the head with this of the computer screen. The other components are used to set and hold the cameras in place. Figure 1(b) shows an anonymous subject with such a headband.

Figure 2 illustrates the data acquisition process. The headband is connected to the host computer, which is connected through an Ethernet link to the display computer. Synchronisation with the image displayed on the screen of the display computer (being looked at by the software engineer) is perform via a provided API, which allows the display computer to control the host computer to synchronise the cameras and the display screen and to generate the data. The generated data is stored on the host computer and transferred to the display computer for analysis at the end of each experiment. The host computer allows experimenters to calibrate the cameras and to control the experiments.

We use such EyeLink II systems to study software engineers' eye movements over the constituents of class diagrams and the dwell time of fixations on individual constituents such as classes, interfaces, and relationships.

2.3 Collected Data

An experiment divides in a set of trials. A trial contains the data collected while a software engineer looks at one image on the display computer, an experiment groups many trials performed successively with (possibly) different images.

The data collected during a trial contains raw coordinates and three types of events: fixations, blinks, and saccades. Each type of event is characterised by the position of the eyes relative to the display screen (two positions for saccades) and by a timestamp in milliseconds relative to the beginning of each trial.

The data collected during one of our typical trials ranges between 200 and 5,000 kilobytes of data, depending on the duration of the experiment. It contains between 10,000 and 200,000 lines of raw data representing 50 and up to 500 fixations, saccades, and blinks. Fixations, saccades, and blinks are intertwined with the raw data on the fly, during data collection.

This data is stored in a binary format known as Eyelink Data Files (EDF). The EDF format is convenient for quick addition of new data and treatment, in particular conversion into ASCII. Table 1 describes the skeleton of an EDF file: An EDF file divides in a header and a body; The header contains information about the experiments, the trial, calibration, drift correction, and the type of recorded events; The body stores both the raw data and events such as beginnings and ends of fixations or saccades.

3 Visualisation Technique

We must aggregate and present the data collected for a same trial across as many experiments as possible to build sound general hypotheses on the acquisition of information by software engineers from class diagrams. We develop a new visualisation technique to aggregate data from several trials and to highlight the parts of a class diagram more or less used by software engineers to acquire information. The main hypothesis of our visualisation technique is that we can equate fixation with attention as suggested for example by Rock [20].

3.1 Previous Work

Previous work attempted to quantify traces of eye-movements rather than parameters such as mean fixation duration, to identify clusters of meaningful fixations, or to manipulate images.

DeCarlo and Santella [2] used eye-trackers to identify the point of attention of a subject on an image and, using this knowledge,

Header	
General information, including the name of the class diagram	MSG 82177 TRIALID PIX1 Normal MSG 82178 !V TRIAL_VAR_DATA Normal MSG 82210 !V IMGLOAD FILL images/UMLDiagram1.jpg
Calibration	MSG 208698 !CAL Calibration points: MSG 208698 !CAL 176.2, 74.5 -808, 3467 ... MSG 230854 !CAL VALIDATION HV9 R RIGHT GOOD ERROR 0.55 avg. 0.97 max OFFSET 0.45 deg. -13.2,5.2 pix.
Drift correction	MSG 253038 DRIFTCORRECT R RIGHT at 640,512 OFFSET 0.83 deg. -20.9,14.5 pix.
Information on the recorded data	MSG 253043 !MODE RECORD P 500 2 1 START 253044 RIGHT SAMPLES EVENTS PRESCALER 1 VPRESCALER 1 PUPIL AREA EVENTS GAZE RIGHT RATE 500.00 TRACKING P FILTER 2 SAMPLES GAZE RIGHT RATE 500.00 TRACKING P FILTER 2
Body	
Eye positions	253048 641.7 512.0 499.0 . ...
Beginning and end of a fixation with in-between samples of eye positions	SFIX R 253052 ... 253310 643.7 508.0 496.0 . ... EFIX R 253052 253310 260 641.5 509.1 504
Beginning and end of a saccade with in-between samples of eye positions	SSACC R 253312 ... 253328 691.9 435.4 522.0 . ... ESACC R 253312 253346 36 644.0 505.0 706.3 394.3 4.43 260
Following beginnings and ends of fixations and saccades with in-between samples of eye positions	SFIX R 253348 ... EFIX R 253348 253610 264 714.8 404.2 452 SSACC R 253612 ...
End	
End of the recording	END 267755 SAMPLES EVENTS RES 32.69 29.12

Table 1: Details of the collected data.

to compute a stylised and abstract version of the image. Their work falls in the field of non-photorealistic rendering. They developed a perceptual model to translate the data gathered from an eye-tracker into predictions about which elements of the image carry information interesting to the subject. Then, they proposed an algorithm to stylise and abstract the image, thus reducing the perceptual and cognitive effort required to understand the image by removing extraneous details. Their work is especially interesting with photographs.

Santella and DeCarlo [21] proposed a robust clustering technique to group sets of fixations either spatially or spatially and temporally. Their technique is robust because its results are not affected by noise and outliers. It can be used to locate areas of interest for a subject or subjects and to quantify the interest in each area. It uses a mean shift procedure which proceeds deterministically without the need to choose a number of clusters in advance. Our visualisation technique performs in the opposite of this technique: We assume the knowledge of the areas of interest and we want to allocate fixations in these areas (to compute percentages) rather than to cluster the fixations.

Wooding [26] introduced the concept of fixation maps. A fixation map is a three-dimensional map in which the two first dimensions are the coordinates of the fixations under study (either from one subject or from several subjects) and the third dimension is a value describing the discrimination, detection, or perception achievable from some considered fixations. The third dimension is deliberately left vague by the author. A fixation map is built by computing for each fixation location the third dimension, for example using a 3-dimensional gaussian or a cylinder. If more than one gaussian overlap at the location of one fixation, their value is added to the third dimension. The width of the gaussian or cylinder depends on the size of the area over which a fixation exists. Using fixation maps, it is thus possible to aggregate fixations and to highlight the areas which received most attention.

Previous work proposed analyses and studied the data collected using eye-tracking systems. Yet, to the best of our knowledge, no previous work proposed a technique to display this data and the amount of attention in areas of interest.

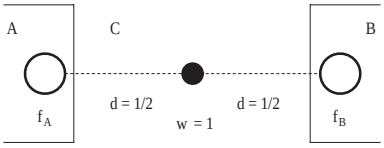


Figure 3: Problem of juxtaposition of fixations (or groups thereof) in fixation maps, where d is a distance and w the weight of a point.

3.2 Areas of Interest

Our technique builds on previous work on clustering and fixation maps and introduces the concept of area of interest. Fixation maps are useful to show the areas of an image that attracted most of the fixations by highlighting these areas through colour gradation. Yet, it combines the data from several trial linearly and, if used to combine several trials, meets the problem of juxtaposition among fixations.

The problem of juxtaposition arises when two fixations (or group thereof) belong to two distinct yet close areas of interest. Figure 3 shows a typical case of problem of juxtaposition: Although fixation f_A pertain to area A and fixation f_B belongs the distinct area B and the two areas are separated by area C, a fixation map would show a continuum of interest in the empty area C because each point in area C, such as the black point, is weighted relatively to the fixations f_A and f_B, with a weight of 1 in the example.

We avoid the problem of juxtaposition when aggregating and presenting data from more than one trial through the use of areas of interest. We define an area of interest as an area in a class diagram that the experimenter considers as relevant to program comprehension activity, excluding pieces of the image not belonging to an area of interest. The experimenter may choose areas of interest before trials or identify these areas after visual analyses of the clustered collected data.

Typically with class diagrams, areas of interest are rectangles circumscribing the graphical representations of classes and interfaces and polygons surrounding relationships (such as in-

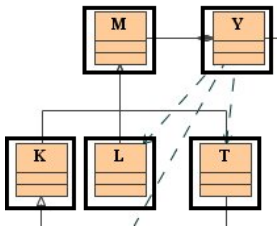


Figure 4: Some areas of interest (black rectangles) in a subset of a typical class diagram.

heritance, implementation, or association relationships). Figure 4 shows some areas of interest in a subset of a class diagram used in our case studies (black rectangles around classes).

We use areas of interest to aggregate fixations and saccades from many trial while highlighting the areas with more or less fixations with respect to one another. For each area of interest, we count the number of fixations and–or saccades in the area across trials and compute percentages with respect to the overall number of fixations and–or saccades. Thus, we associate a percentage with each area of interest and can show differences among areas and thus highlight which areas received more of less attention than others without the juxtaposition problem.

The concept of areas of interest is simple and can be applied to any type of image where experimenters can identify such areas.

3.3 The Taupe Data Viewer

We have implemented our visualisation technique in the Taupe data viewer. The Taupe data viewer (*Thoroughly Analysing the Understanding of Programs through Eyesight*[1]) is implemented 100% in Java. It uses the API provided by SR Research to handle EDF files directly in Java. It is a simple program consisting of 20 classes and 130 methods for 1,500 lines of commented codes (excluding comments and libraries).

Figure 5 shows the user interface of the Taupe data viewer. Part A displays the class diagram shown to software engineers during some trials. Part B is used to load the data viewer data from experiments, choose the trials of interest in these experiments, show/hide fixations and–or saccades, and include/exclude fixations and–or saccades from the computation of the percentages of the areas of interest. Part C is used to load sets of areas of interest into the data viewer, to display these areas, and to shift the position of the collected data to compensate any drift during calibration of the eye-trackers.

[1]Taupe means *mole* in French.

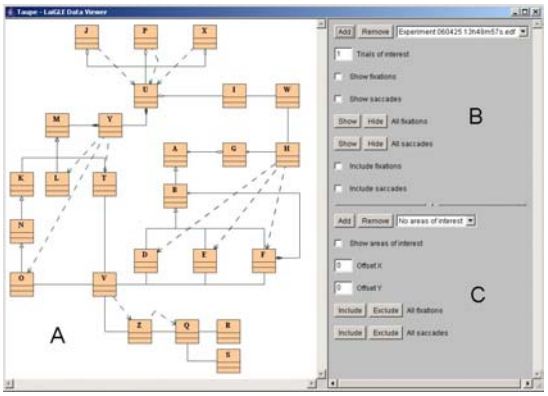


Figure 5: User interface of the Taupe data viewer and its three parts.

The Taupe data viewer uses alpha composition to display fixations, saccades, and areas of interest over the class diagram without cluttering the diagram while differentiating areas which received much attention from those which did not.

Figure 6(b) shows the fixations from several software engineers and the computed areas of interest superimposed over the typical class diagram used in our first case study and shown in Figure 6(a). Dots represent fixations, some parts of the class diagram are almost invisible to indicate that they did not receive attention from the software engineers while others (such as `F`) received much attention. Figures 6(a) and 6(b) show that, using our data viewer, we can accurately highlight the parts of a class diagram that received much attention from software engineers during program comprehension.

4 Case Studies

We perform two case studies to highlight the use of eye-tracking systems to understand the program comprehension activity better. In

these case studies, we show two typical class diagram to software engineers and ask, for each, a typical question that could arise during the development or the maintenance of the described programs. We obtain surprising results on the use of relationships.

4.1 Experimental Setting

We want to understand how software engineers use concretely class diagrams during program comprehension activities. Thus, the subjects of our experiments are software engineers and the objects are class diagrams representing some programs. In the perspective of generalising the results of our experiments, we must choose the subjects and objects carefully.

Subjects. Subjects should be representative of software engineers *in general* and be sufficient in number to account for incidental variations. We use subjects from a convenient sample consisting of graduate students in software engineering at the Department of Informatics and Operations Research at University of Montreal to perform the first experiments reported in the following. Graduate students have an reasonable knowledge of UML in general and class diagrams in particular, with respect to software engineers *in general*. Although most do not have industrial experience in development and maintenance, they all have designed and developed small-scale programs using UML during their studies. We perform our experiments with a dozen subjects to leverage further variations.

Objects. The objects of our experiments are class diagrams modelling object-oriented programs. We use two different class diagrams from two different programs to avoid reinforcement learning [19, 25] after the first set of trials. The first class diagram, CD1 in Figure 6(a), models a subset of the PTIDEJ program [6]. We render this class diagram anonymous to avoid bias as some subjects participated in its development. The second class diagram, CD2 in Figure 7(a), models a simulation of an ATM machine written by Russell C. Bjork and freely available at `http://www.math-cs.gordon.edu/courses/cs211/ATMExample/`. We use this example because it provides a complete yet manageable class diagram.

Questions. We ask the subjects one question per class diagram to trigger the program comprehension activity. The question associated to CD1 concerns the design of the modelled program and the addition of a new class in the class hierarchy):

> Q1: Where to add a class `Other` such that its instances belong to a tree of objects with instances of class `F` as containers?[2]

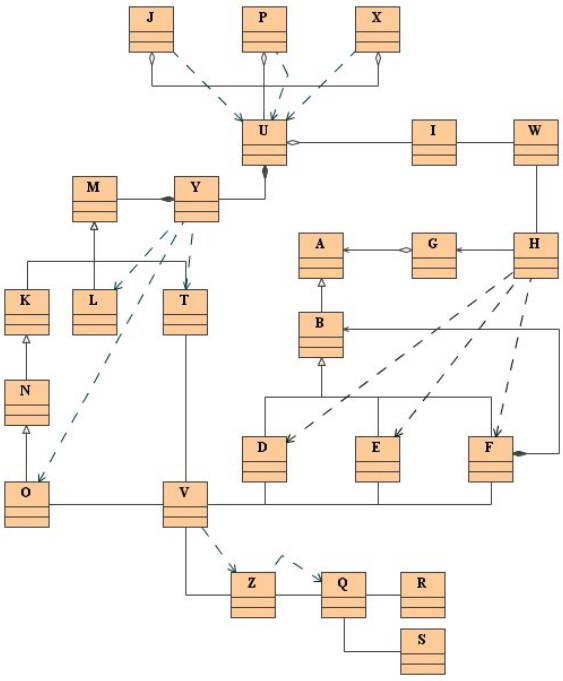
The second question related to CD2 also concerns design and class hierarchy but involved changing the existing class hierarchy:

> Q2: How to add a new type of `Transaction` that is not associated with a `Session`?[2]

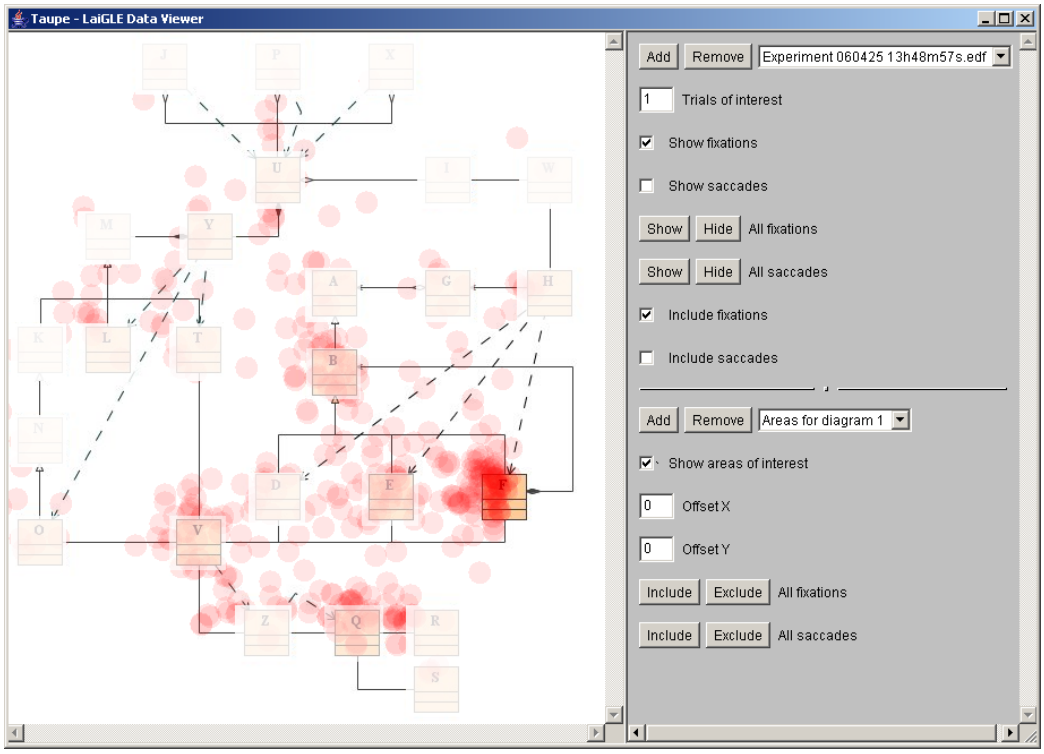
Recording. We ask subjects to come for the experiments in sequential order. We monitored in real-time and continuously the recordings to prevent drift and errors in the measures. We asked subjects not to communicate on the experiments with future subjects. We record the subjects' eye movements, including events related to fixations and saccades. We do not record events related to blinks because we do not believe they are relevant to understand the program comprehension activity. Results of the experiments are kept strictly confidential through the automatic generation of the names of the EDF files and the automatic renaming of all the files in random order to prevent identification through time stamps. We ensure that subjects understood the question and perform the expected program comprehension activity by analysing their answers to the questions. In our experiments, no subject has been rejected because of misunderstanding.

Results. Figures 6(b) and 7(b) show screenshots of the TAUPE data viewer showing the results of the experiments for CD1 and CD2, respectively. As expected, the collected data

[2]The questions are translated from French.

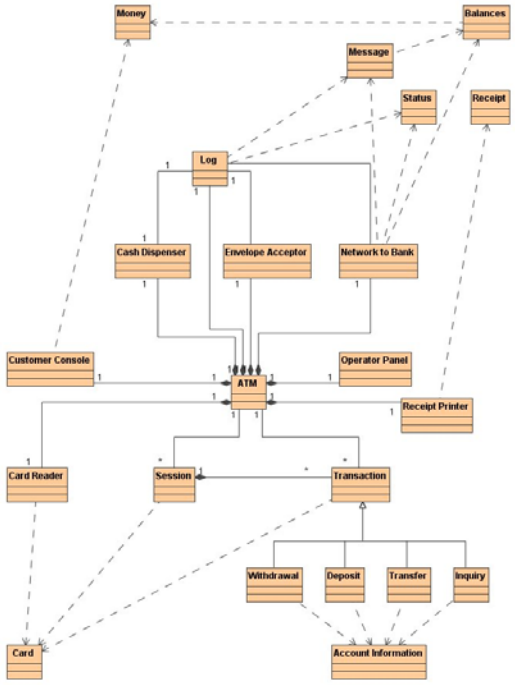


(a) A typical class diagram (CD1).

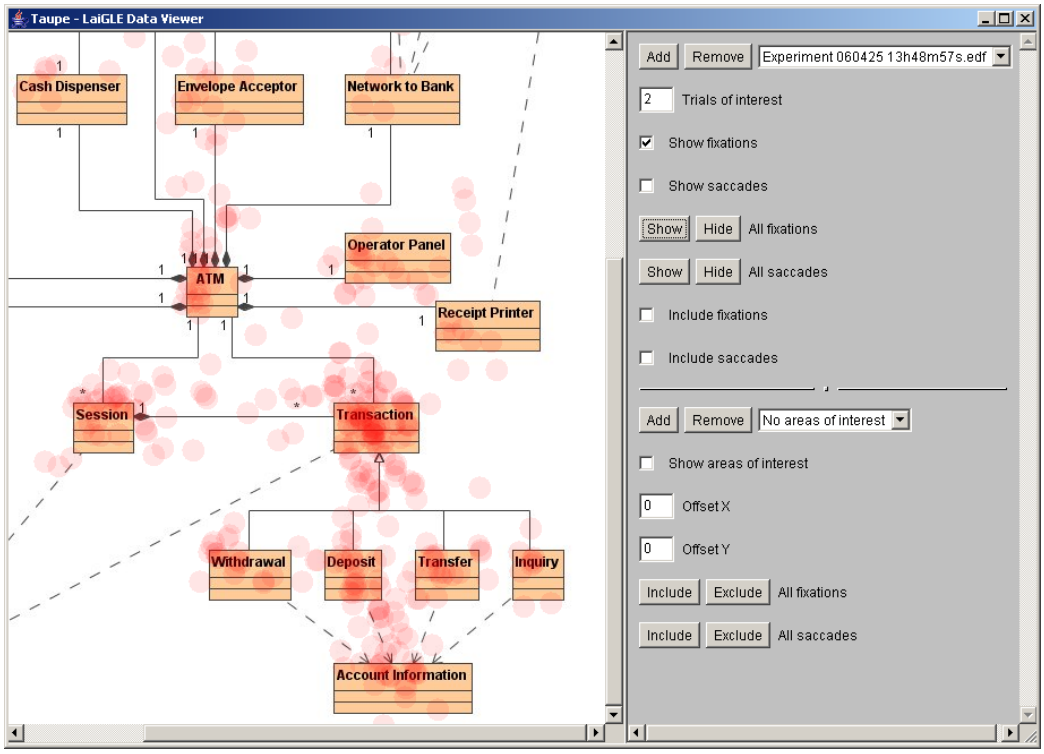


(b) Results of the analysis of the class diagram with fixations and areas of interest.

Figure 6: Analysis of the typical class diagram CD1



(a) Class diagram of an ATM machine (CD2).



(b) Fixations on the class diagram without areas of interest.

Figure 7: Analysis of the typical class diagram CD2

supports the idea that software engineers first browse the class diagrams seemingly randomly to identify most useful parts and then focus on these parts for the program comprehension activity at hand (to answer the questions).

Thus, it is not surprising to observe more fixations towards the classes most useful:

- CD1: Software engineers concentrated their attention on class `F` and related classes, in particular `B`, `D`, and `E`[3].
- CD2: Software engineers fixed their attention on class `Transaction`, its subclasses (such as `Withdrawal`), and class `Session`[4].

Surprisingly, however, software engineers do not seem to follow binary class relationships, such as inheritance and composition. In none of the experiments, either using the Taupe data viewer to display fixations and saccades or SR Research default viewer to replay the eye movements could we notice software engineers following the relationships among classes.

This is a very surprising result, in particular in face of the several techniques existing in the literature to choose correct relationships among classes, such as [9, 14], and to recover binary class relationships from source code, such as [7, 10]. We hypothesise that we obtain these results because the class diagrams are quite simple and the questions do not require *per se* the use of the binary class relationships. Yet, we expected software engineers to confirm their findings using the relationships among classes.

Threats to Validity. The previous results are subject to some threats to validity:

- Time thresholds, peripheral vision, and offsets: it is unlikely that we missed fixations, in particular on the binary class relationships, due to the thresholds used to identify fixations from raw data, to the use of peripheral vision by software engineers while focusing on classes, or to measurement problems. Yet, further (different) experiments are required to confirm our findings. In particular, we will study other diagrams with different layouts where entities are further apart.
- Subjects: we used a convenient sample of graduate students to perform the experiments. We must perform again our experiments with other subjects to level any particularity of our convenient sample.
- Questions: the question used to trigger the program comprehension activity are quite simple and should be further refined to better frame the software engineers' behaviour.

Despite these threats to validity, the reported results are important because they open interesting research avenues to better understand the use of class diagrams during program comprehension and provide a base on which to build further experiments.

5 Conclusion

We applied eye-tracking to a first study of the use of class diagrams by software engineers during program comprehension. We report interesting results from two case studies on the apparent lack of use of the relationships among classes. Our first experiments—although their validity should be further confirmed—are only a beginning in promising research to better understand program comprehension.

This work is part of the on-going Taupe project being carried out at the Department of Informatics and Operations Research of University of Montreal. It is but a first step towards a better understanding of the use of class diagrams and other software artifacts during program comprehension. Future work includes:

- Performing more case studies with different and more complex class diagrams and other types of diagram [17], different activities of program comprehension, and with other participants, including software engineers from the industry.
- Evaluating the different strategies used by software engineers to acquire information from class diagrams with respect to the activity of program comprehension at hand.

[3] Adding `Other` as a subclass of `B` answers Q1.

[4] The hierarchy of `Transaction` must be modified to add an extra `ManagedTransaction` class to answer Q2.

- Studying other sources of information such as UML sequence diagrams, source code, adjacency matrices [5], and other visualisation techniques [12], such as in [8, 13].
- Formulating hypotheses, based on the case studies and their results, on the concrete use of information by software engineers and developing a theory of program comprehension to help in developing and in evaluating new visualisation techniques.

Acknowledgements

The author thanks gratefully the Canadian Foundation for Innovation for the financial support, SR Research for the their invaluable help, Pierre Poulin for the many interesting discussions, and the several participants to the case studies.

References

[1] Helen C.Purchase, Jo-Anne Allder, and David Carrington. Graph layout aesthetics in UML diagrams: User preferences. *journal of Graph Algorithms and Applications*, 6(3):255–279, June 2002.

[2] Doug DeCarlo and Anthony Santella. Stylization and abstraction of photographs. In Tom Appolloni, editor, *proceedings of the* 29^{th} *conference on Computer graphics and interactive techniques*, pages 769–776. ACM Press, July 2002.

[3] Holger Eichelberger. Nice class diagrams admit good design? In John T. Stasko, editor, *proceedings of the* 1^{st} *symposium on Software Visualization*, pages 159–168. ACM Press, June 2003.

[4] Erich Gamma and Kent Beck. Test infected: Programmers love writing tests. *Java Report*, 3(7):37–50, July 1998.

[5] Mohammad Ghoniem, Jean-Daniel Fekete, and Philippe Castagliola. A comparison of the readability of graphs using node-link and matrix-based representations. In Matt Ward and Tamara Munzner, editors, *proceedings of the* 10^{th} *symposium on Information Visualisation*, pages 17–24. IEEE Computer Society Press, October 2004.

[6] Yann-Gaël Guéhéneuc. A reverse engineering tool for precise class diagrams. In Janice Singer and Hanan Lutfiyya, editors, *Proceedings of the* 14^{th} *IBM Centers for Advanced Studies Conference*, pages 28–41. ACM Press, October 2004.

[7] Yann-Gaël Guéhéneuc and Hervé Albin-Amiot. Recovering binary class relationships: Putting icing on the UML cake. In Doug C. Schmidt, editor, *Proceedings of the* 19^{th} *conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 301–314. ACM Press, October 2004.

[8] Irit Hadar and Orit Hazzan. On the contribution of UML diagrams to software system comprehension. *journal of Object Technology*, 3(1):143–156, January–February 2004.

[9] Brian Henderson-Sellers and Franck Barbier. A survey of the UML's aggregation and composition relationships. *L'objet : Logiciel, Base de données, Réseaux*, 5(3/4):339–366, December 1999.

[10] Daniel Jackson and Allison Waingold. Lightweight extraction of object models from bytecode. In David Garlan and Jeff Kramer, editors, *proceedings of the* 21^{st} *International Conference on Software Engineering*, pages 194–202. ACM Press, May 1999.

[11] Kostas Kontogiannis. ICPC web-site, April 2006. `http://www.icpc2006.uwaterloo.ca/`.

[12] Guillaume Langelier, Houari A. Sahraoui, and Pierre Poulin. Visualization-based analysis of quality for large-scale software systems. In Tom Ellman and Andrea Zisma, editors, *proceedings of the* 20^{th} *internatinal conference on Automated Software Engineering*. ACM Press, November 2005.

[13] Gail C. Murphy, Mik Kersten, Martin P. Robillard, and Davor Čubranić. The emergent structure of development tasks. In Andrew P. Black, editor, *proceedings of the* 19^{th} *European Conference on Object-Oriented Programming*, pages 33–48. Springer-Verlag, July 2005.

[14] James Noble and John Grundy. Explicit relationships in object-oriented development. In Bertrand Meyer, editor, *proceedings of the* 18^{th} *conference on the Technology of Object-Oriented Languages and Systems*, pages 211–226. Prentice-Hall, November 1995.

[15] Object Management Group. *UML v1.5 Specification*, March 2003.

[16] Stephen E. Palmer. *Vision Science: Photons to Phenomenology*. The MIT Press, 1^{st} edition, May 1999.

[17] Ivan P. Paltor and Johan Lilius. Digital sound recorder: A case study on designing embedded systems using the UML notation. Technical Report TR-234, Turku Centre for Computer Science, January 1999.

[18] Dewayne E. Perry and Alexander L. Wolf. Foundations for the study of software architecture. *Software Engineering Notes*, 17(4):40–52, October 1992.

[19] Václav Rajlich. Program comprehension as a learning process. In Yingxu Wang, editor, *proceedings of the* 1^{st} *International Conference on Cognitive Informatics*, pages 343–347. IEEE Computer Society Press, August 2002.

[20] Irvin Rock and Daniel Gutman. The effect of inattention on form perception. *Journal of Experimental Psychology: Human Perception and Performance*, 7:275–285, 1981.

[21] Anthony Santella and Doug DeCarlo. Robust clustering of eye movement recordings for quantification of visual interest. In Andrew Duchowski and Roel Vertegaal, editors, *proceedings of the* 3^{rd} *symposium on Eye Tracking Research and Applications*, pages 27–34. ACM Press, March 2004.

[22] Jochen Seemann. Extending the Sugiyama algorithm for drawing UML class diagrams: Towards automatic layout of object-oriented software diagrams. In Giuseppe Di Battista, editor, *proceedings of the* 5^{th} *international symposium on Graph Drawing*, pages 415–424. Springer-Verlag, September 1997.

[23] Elliot Soloway. Learning to program = Learning to construct mechanisms and explanations. *Communications of the ACM*, 29(9):850–858, September 1986.

[24] Dabo Sun and Kenny Wong. On evaluating the layout of UML class diagrams for program comprehension. In James R. Cordy and Harald Gall, editors, *proceedings of the* 13^{th} *International Workshop on Program Comprehension*, pages 317–326. IEEE Computer Society Press, May 2005.

[25] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1^{st} edition, March 1998.

[26] David S. Wooding. Fixation maps: Quantifying eye-movement traces. In Roel Vertegaal and John W. Senders, editors, *proceedings of the* 2^{nd} *symposium on Eye Tracking Research and Applications*, pages 31–36. ACM Press, March 2002.